



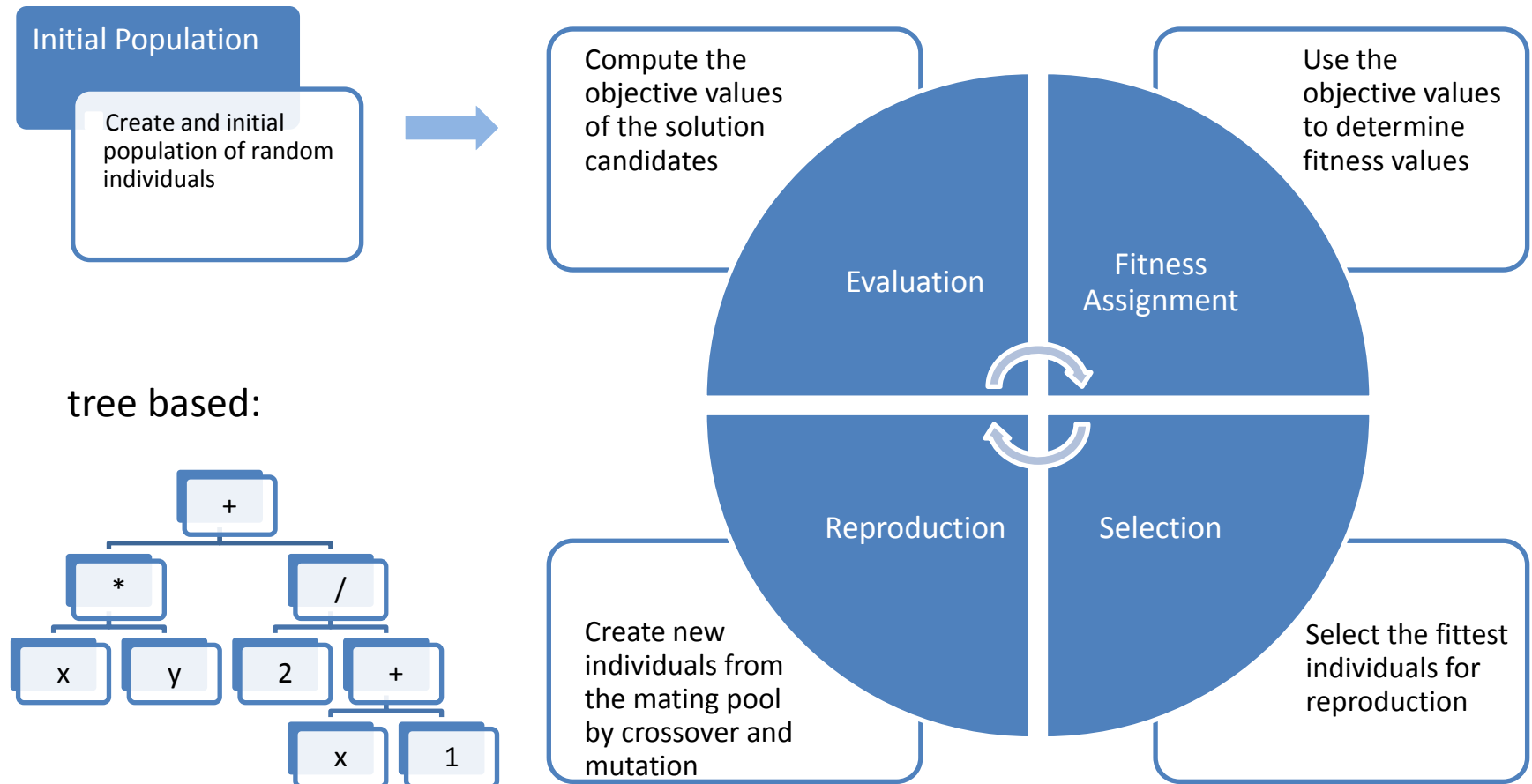
Novel Loop Structures and the Evolution of Mathematical Algorithms

Mingxu Wan, Thomas Weise, Ke Tang

EuroGP2011

- Introduction
- Research Goal
- Loop Representations
- Problems in GP
- Loop Representations (Novel)
- Experimental Setup
- Comparison
- Category
- Loop Utilization
- Interesting Equation
- Conclusion

GP Evolutionary Cycle:



- GP method could solve: permutation, searching, quantum computing, electronic design, game playing, sorting...
- rarely used in: automated programming
- It is not easy to synthesize mathematical algorithms with GP.
- most of research before only aim to special problems.
- In our research, we want to answer the question: Is there such a way that could solve synthesizing mathematical algorithms both precisely and effectively in general?

In our research we want to solve:

- deterministic algorithms
- discrete results and variables

Such as:

- Polynomial Problem
- GCD Problem
- Factorial Problem
- Prime determination

...

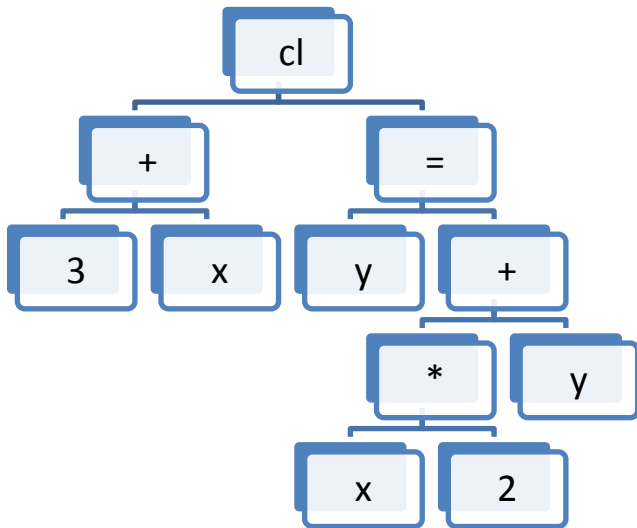
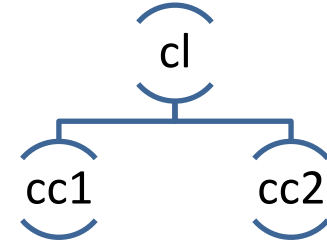
In order to solve such problems, we need:

- Memory
- Loop
- Recursion, Function call, ...

Loop Representations

1. Counter Loop (CL)

- two children cc1 and cc2.
- cc1 is the number of iteration.
- cc2 is the loop body.



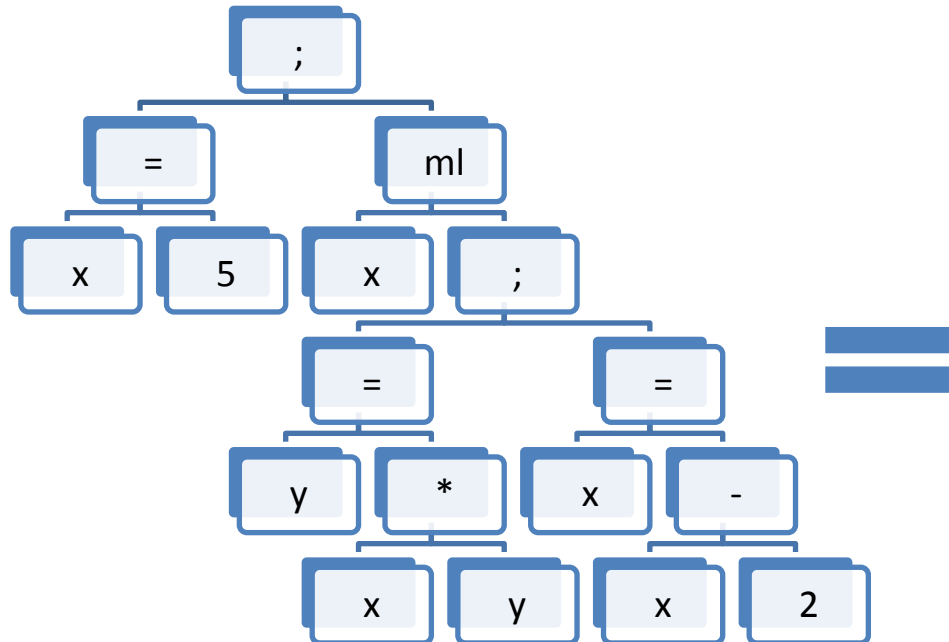
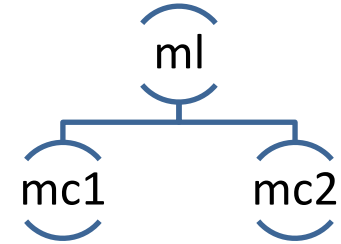
=

```
for(i=0; i<x+3; ++i) {
    y+=2*x;
}
```

Loop Representations

2. Memory Loop (ML)

- two children mc1 and mc2.
- mc1 is a variable and decreased by one in each loop iteration
- mc2 represents the loop body, executed until mc1 becomes less or equal 0.

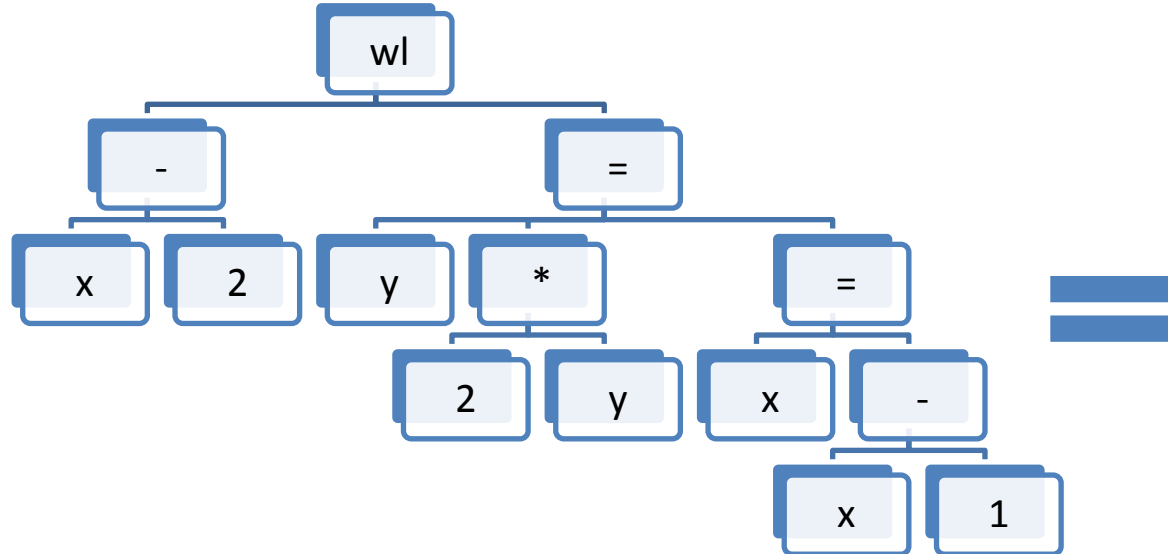
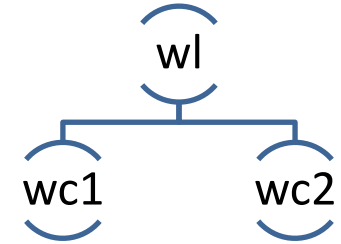


```
for (x=5; x>0; --x)
{
    y=y*x;
    x=x-2;
}
```

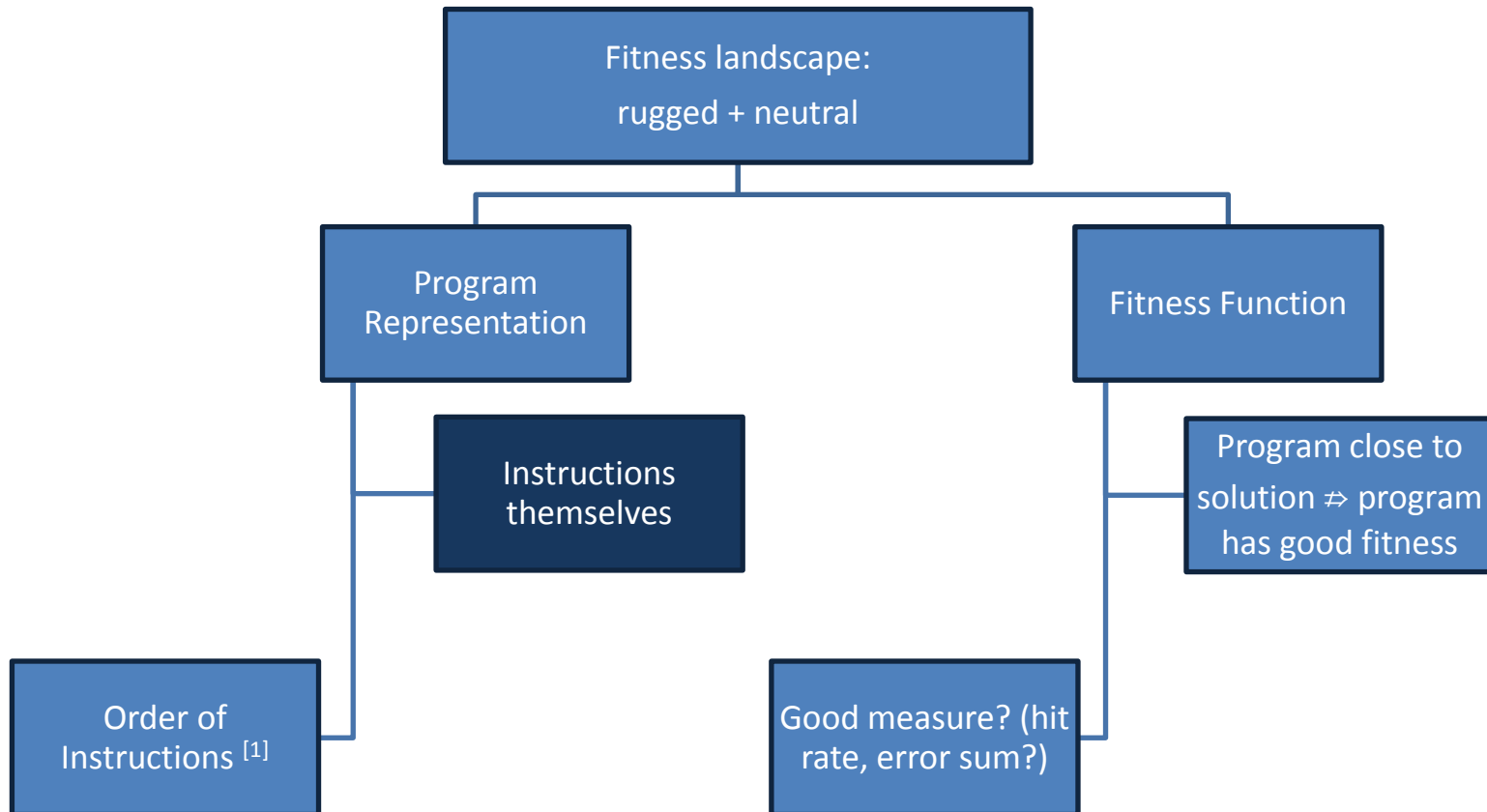
Loop Representations

3. While Loop (WL)

- two children wc1 and wc2.
- wc1 represents the loop condition.
- wc2 is repeatedly evaluated until wc1 becomes 0.



```
while (x!=2) {
    y=y*2;
    x=x-1;
}
```

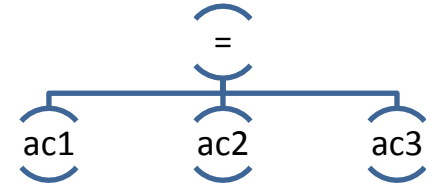


[1] Thomas Weise: "Evolving Distributed Algorithms with Genetic Programming," PhD Thesis, May 4, 2009 at University of Kassel,

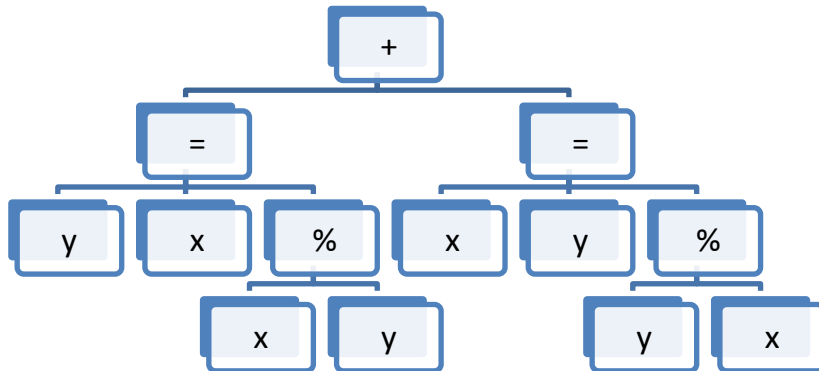
1. We want to move away from explicitly defined iteration numbers and explicit conditions
2. Instead, want loops where the number of iterations implicitly results from the loop body
3. We aim to prevent useless iterations

Loop Representations (Novel)

4. Conditional Assignment (CA)



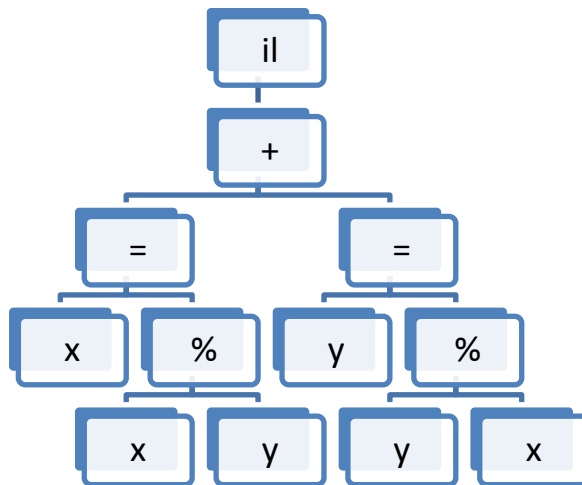
- three children, ac1, ac2, and ac3.
- ac1 is a condition expression.
- ac2 is the target variable.
- ac3 will be assigned to the ac2 if ac1 evaluates to a non-zero value.
- program tree will be evaluated repeatedly until no variable changes anymore.



x	y
5	3
2	1
0	1
0	1

5. Implicit Loop (IL)

- a single child ic1.
- ic1 represents the loop body.
- evaluated until there is no change in any variable.



x	y
15	9
6	3
0	3
0	3

The basic operators which was used in our GP method to synthesize algorithms:

Operator	Function
$+, -, *$	default arithmetic operators
$a \% b$	protected modulo division, returns $a \bmod b$ if $b \neq 0$, and a otherwise
$a = b$	set the value of variable a to the result of the expression b (which is also returned)
$a; b$	sequential execution of two expressions a and b
v_1, v_2, \dots, v_n	the value of the one of n memory variables
0,1	the only two ephemeral random constants available

For each training cases t_i , after executing an evolved program P , we expect its result, $P(t_i)$ to be stored in its last memory cell v_n , $n = 2$.

First, we propose a trivial polynomial problem

$$\Phi_1(t_i) = t_i^3 + t_i 2 + 2 * t_i$$

We use $tc = 100$ training cases $i = 1, 2, \dots, 100$ and, at the beginning of each program execution, initialize all variables with t_i .

The sum problem

$$\Phi_2(t_i) = \sum_{j=1}^{t_i} j$$

The GCD problem

$$\Phi_3(t_{i,1}, t_{i,2}) = \text{gcd}(t_{i,1}, t_{i,2})$$

In the memory loop representation ML, each possible solution requires at least three memory cells so we allowed this representation to utilize $n = 3$ variables (initially $v_3 = 0$).

The factorial problem

$$\Phi_4(t_i) = t_i!$$

In our experiments, we used the Genetic Programming implementation of the ECJ framework:

- Population size = 1000
- Generation limit = 100
- Tournament selection contestants = 7
- Point mutation = 10%
- Sub tree exchange crossover=90%
- Maximum tree depth=17
- 100 independent runs for each configuration

$$f_1(P) = \sum_{i=1}^{tc} |\Phi_{pr}(t_i) - P(t_i)| \quad (1)$$

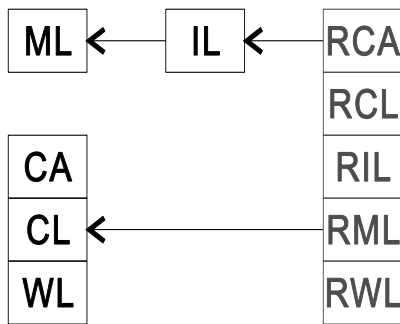
$$f_2(P) = |\{(i \in 1 \dots tc) \cap (\Phi_{pr}(t_i) = P(t_i))\}| \quad (2)$$

$$f_3(P) = 1 \quad (3)$$

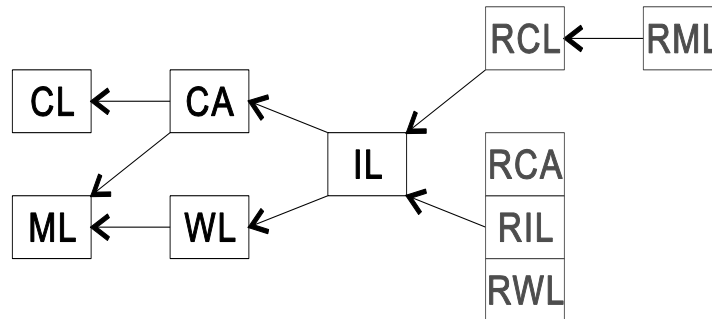
f_3 turns GP into random work. R is prepended to the names of the random walk versions of the experiments.

Comparison

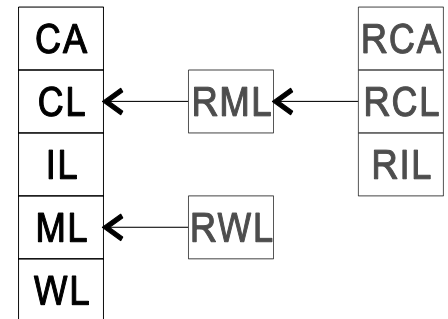
- Comparison with Random Walks
- Statistical comparisons: Mann-Whitney U test ($\alpha = 0.02$, 2-tailed)



factorial(f_2)



sum(f_1)



GCD(f_1)

As experiments shown that the programs generated by the same representation also could have different degrees of loop effectivity:

```
v1∈1...10; //input  
V2=5; //output (init)  
While (v1>3) {  
    v2=v2;  
    V1--;  
}
```



Category A

There is no loop instruction in the program that influences the result in any way.

Category B

At least one loop is present and is an effective loop, its body is executed more than once and the result of the program depends on the number of executions.



```
v1∈1...10; //input  
V2=5; //output (init)  
While (v1<5) {  
    V2++;  
    V1++;  
}
```

As experiments shown that the programs generated by the same representation also could have different degrees of loop effectivity:

Category C

At least one loop which has degenerated to an alternative in the style of an if-then rule.



```
v1∈1...10; //input  
v2=5; //output (init)  
While (v1<5) {  
    v2=3;  
    v1++;  
}
```

```
v1∈1...10; //input  
v2=5; //output (init)  
While (v1>0) {  
    v2++;  
    v1++;  
}
```



Category D

One endless loop is present which exceeds maximum number of allowed iterations

Loop Utilization

Problem		A	B	C	D
CL	f_1	60/60	0/0	40/40	0/0
CL	f_2	33/72	1/28	0/0	1/25
RCL	f_3	0/100	0/0	0/0	0/0
ML	f_1	90/90	10/10	0/0	0/0
ML	f_2	22/72	3/24	0/0	0/9
RML	f_3	0/91	0/9	0/0	0/0
WL	f_1	90/92	0/1	0/0	0/9
WL	f_2	91/91	0/9	0/0	0/9
RWL	f_3	0/99	0/1	0/0	0/1
CA	f_1	0/0	100/100	0/0	0/0
CA	f_2	59/59	41/41	0/0	0/0
RCA	f_3	0/69	0/31	0/0	0/1
IL	f_1	56/56	44/44	0/0	13/13
IL	f_2	100/100	0/0	0/0	0/0
RIL	f_3	0/62	0/38	0/0	0/5

Polynomial Problem ($pr = 1$)

Problem		A	B	C	D
CL	f_1	0/2	0/97	0/1	0/96
CL	f_2	0/53	1/47	0/0	0/42
RCL	f_3	0/99	0/1	0/0	0/0
ML	f_1	0/4	26/96	0/0	0/14
ML	f_2	0/46	27/54	0/0	0/14
RML	f_3	0/95	0/5	0/0	0/0
WL	f_1	0/77	0/22	0/1	0/23
WL	f_2	0/85	0/15	0/0	0/15
RWL	f_3	0/98	0/2	0/0	0/2
CA	f_1	0/0	0/100	0/0	0/0
CA	f_2	0/16	1/84	0/0	0/0
RCA	f_3	0/70	0/30	0/0	0/5
IL	f_1	0/5	0/95	0/0	0/0
IL	f_2	0/26	0/74	0/0	0/1
RIL	f_3	0/70	0/30	0/0	0/5

Sum Problem ($pr = 2$)

Loop Utilization

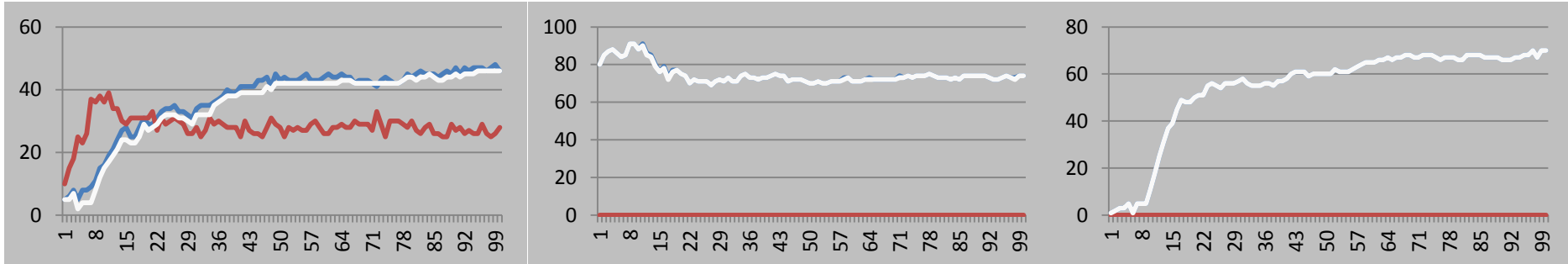
Problem		A	B	C	D
CL	f_1	0/29	0/46	0/25	0/46
CL	f_2	29/83	0/17	0/0	0/17
RCL	f_3	0/83	0/17	0/0	0/17
ML	f_1	0/26	0/74	0/0	0/74
ML	f_2	29/89	1/11	0/0	1/11
RML	f_3	0/93	0/7	0/0	0/7
WL	f_1	0/31	0/69	0/0	0/69
WL	f_2	30/85	5/15	0/0	0/10
RWL	f_3	0/97	0/3	0/0	0/3
CA	f_1	0/0	9/100	0/0	0/7
CA	f_2	0/0	7/100	0/0	0/7
RCA	f_3	0/62	0/38	0/0	0/2
IL	f_1	0/0	1/100	0/0	0/13
IL	f_2	0/1	86/99	0/0	0/0
RIL	f_3	0/54	0/46	0/0	0/2

GCD Problem ($pr = 3$)

Problem		A	B	C	D
CL	f_1	0/2	0/92	0/6	0/90
CL	f_2	0/35	0/45	0/0	0/39
RCL	f_3	0/99	0/1	0/0	0/0
ML	f_1	0/25	14/75	0/0	0/45
ML	f_2	0/37	0/63	0/0	0/40
RML	f_3	0/92	0/8	0/0	0/0
WL	f_1	0/77	0/23	0/0	0/23
WL	f_2	0/94	0/6	0/0	0/6
RWL	f_3	0/100	0/0	0/0	0/0
CA	f_1	0/0	0/100	0/0	0/1
CA	f_2	0/14	0/86	0/0	0/1
RCA	f_3	0/63	0/37	0/0	0/3
IL	f_1	0/13	0/87	0/0	0/0
IL	f_2	0/72	0/28	0/0	0/1
RIL	f_3	0/59	0/41	0/0	0/4

Factorial Problem ($pr = 4$)

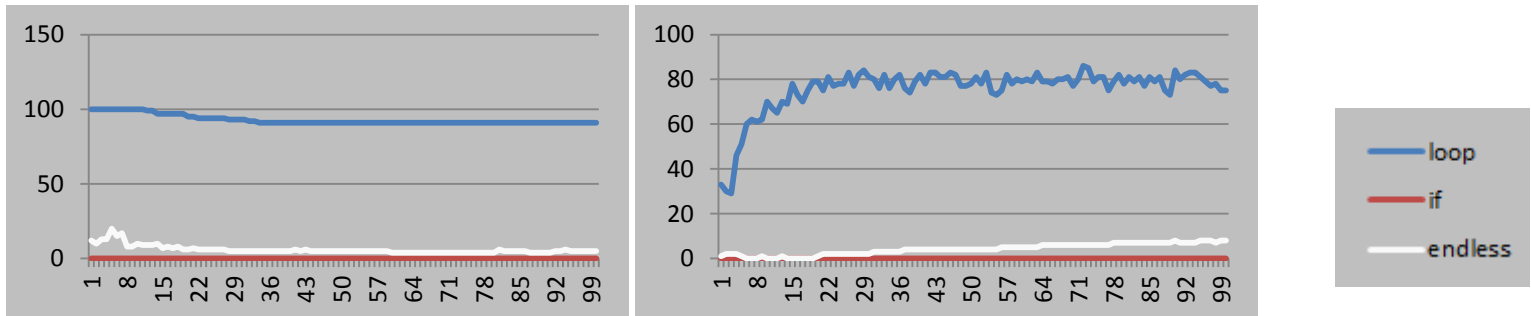
The development of loop utilization



Counter Loop (CL)

Memory Loop (ML)

While Loop (WL)



Conditional Assignment (CA)

Implicit Loop (IL)

Interesting Equation

In the sum problem, we omit the division operator from the instruction set, thus forcing Genetic Programming to synthesize suitable loops.

($\frac{n(n+1)}{2}$ cannot be discovered with the available instructions).

However, we found this:

$$\sum_{k=1}^n k = (2x^4 + 2x^3) \% (4x^2 - 1)$$

Which means that the sum problem could be solved without loops.

- GP makes efficient use of the information provided by the objective functions.
- For non-trivial algorithm synthesis problems, Genetic Programming rarely finds suitable solutions.
- Imperative Standard Genetic Programming in its current form does not perform well in the evolution of non-trivial and non-approximative algorithms.
- Implicit stopping criteria for loops, based on convergence of variables rather than explicit expressions, can seemingly more easily be utilized by Genetic Programming.



Thank you